Open-Source Projects:
Machine Learning for Transportation Data Imputation and Prediction

Reproducible Research Workshop

TRB 103rd Annual Meeting · Washington, D.C., USA

Nicolas Saunier & Xinyu Chen

Polytechnique Montreal, Canada

January 11, 2024

Open-source & reproducible research:

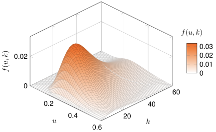
1. GitHub: https://github.com/xinychen
2. Slides: https://xinychen.github.io/slides/transdim.pdf
3. Project website: https://spatiotemporal-data.github.io

ML algorithms

transdim

(1.1k stars)

Visualization tools



awesome-latex-drawing

(1.2k stars)

Other projects at Polytechnique Montreal in video processing for user behavior and safety analysis https://trafficintelligence.confins.net

1. Motivation

2. Storytelling with Data

3. Spatiotemporal Traffic Data Modeling
   - Reformulate traffic data imputation
   - Reformulate traffic forecasting

4. Python Implementation
   - Tools & packages
   - Traffic data processing
   - Switch from CPU to GPU

5. “Open-Source”? Post Something That Matters

Why?

Academia:

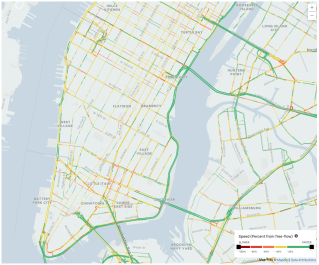
- Open research environment (w.r.t. our team & followers)
- Push by funding agencies and academic institutions: "Research Data Management Policy"
- Interact with researchers from different fields
- Provide platform and benchmark for comparison
- Stimulate new algorithmic ideas

Industry:

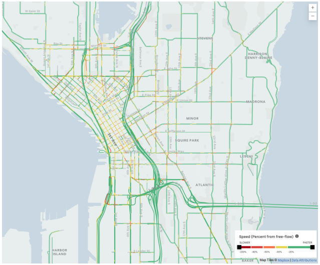
- Provide solutions to some realistic data problems
- Easy to implement and produce results

Storytelling with Data

- Uber (hourly) movement speed data



NYC movement



Seattle movement

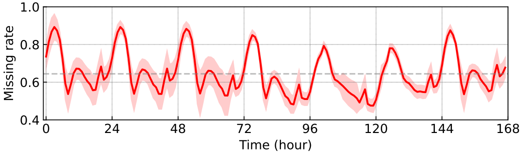
- {road segment, time step (hour), average speed}
- $\boldsymbol{Y} \in \mathbb{R}^{N \times T}$ with N spatial locations $\times$ T time steps
- Computing hourly speed: Road segments have 5+ unique trips.

Issue: Insufficient sampling of ridesharing vehicles on the road network!

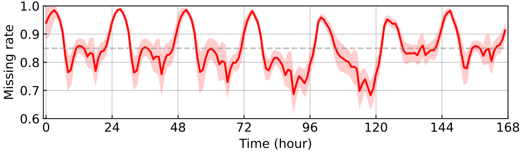
Storytelling with Data

High-dimensional & sparse

- **NYC** movement speed data (2019)
  - 98,210 road segments & 8,760 time steps (hours)
  - Overall missing rate: 64.43%



- **Seattle** movement speed data (2019)
  - 63,490 road segments & 8,760 time steps (hours)
  - Overall missing rate: 84.95%



- How to quantify data quality? Address sparsity?

Reformulate Traffic Data Imputation

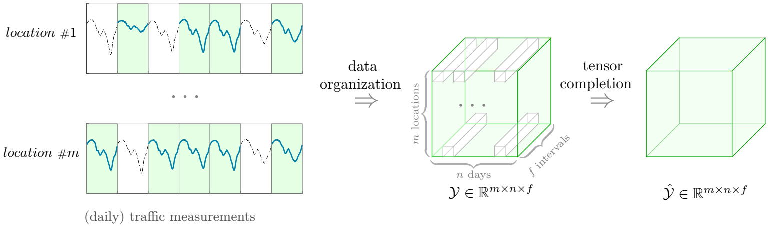
Estimation:

- Imputation & interpolation & forecasting

Imputing missing traffic data

- Tensor completion (Observed index set Ω)

$$\underbrace{\mathcal{P}_{\Omega}(\boldsymbol{\mathcal{Y}})}_{\text{Partially observed}} \xrightarrow{\text{Estimate}} \underbrace{\mathcal{P}_{\Omega}^{\perp}(\boldsymbol{\mathcal{Y}})}_{\text{Unobserved}}$$

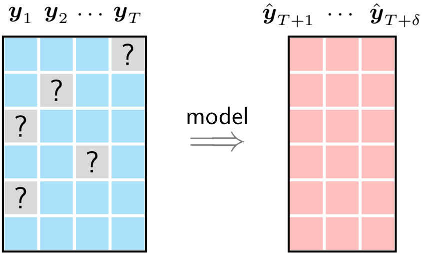


Reformulate Traffic Forecasting

Forecasting urban traffic states with sparse data

- Problem definition (δ-step ahead forecasting)

$$\underbrace{\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T\}}_{\text{Current traffic states}} \xrightarrow{\text{Estimate}} \underbrace{\{\hat{\boldsymbol{y}}_{T+1}, \hat{\boldsymbol{y}}_{T+2}, \ldots, \hat{\boldsymbol{y}}_{T+\delta}\}}_{\text{Future traffic states}}$$



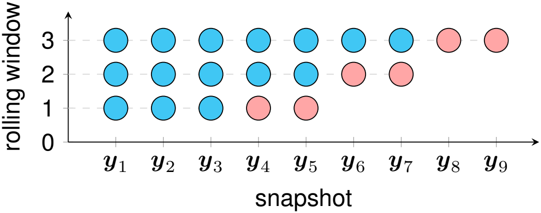
Reformulate Traffic Forecasting

(Rolling) Forecasting urban traffic states with sparse data

1st rolling step: $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3\} \rightarrow \{\boldsymbol{y}_4, \boldsymbol{y}_5\}$

2nd rolling step: $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_4, \boldsymbol{y}_5\} \rightarrow \{\boldsymbol{y}_6, \boldsymbol{y}_7\}$

3rd rolling step: $\underbrace{\{\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3, \boldsymbol{y}_4, \boldsymbol{y}_5, \boldsymbol{y}_6, \boldsymbol{y}_7\}}_{\text{Current traffic states}} \rightarrow \underbrace{\{\boldsymbol{y}_8, \boldsymbol{y}_9\}}_{\text{Future traffic states}}$



Python Implementation

Tools & Packages

Python	CPU computing	GPU computing
jupyter	NumPy	CuPy
colab	pandas	*NumPy for GPU

Python Implementation

Traffic Data Processing

- Data format: `.npz` (compressed format)
- Easy to use
  - Connect with `numpy` (for **CPU**)
  - Connect with `cupy` (for **GPU**)

NYC Uber movement dataset[1]:

- e.g., `hourly_speed_mat_2019_1.npz` (91 MB)
  - 98210×744 matrix
  - 23,228,581 observations
  - Whole month of January 2019

[1] https://github.com/xinychen/tracebase

Python Implementation

Pre-process several open datasets[2]

Open data

In this project, we have adapted some publicly available data sets into our experiments. The original links for these data are summarized as follows,

- **Multivariate time series**
  - Birmingham parking data set
  - California PeMS traffic speed data set (large-scale)
  - Guangzhou urban traffic speed data set
  - Hangzhou metro passenger flow data set
  - London urban movement speed data set (other cities are also available at Uber movement project)
  - Portland highway traffic data set (including traffic volume/speed/occupancy, see data documentation)
  - Seattle freeway traffic speed data set
- **Multidimensional time series**
  - New York City (NYC) taxi data set
  - Pacific surface temperature data set

For example, if you want to view or use these data sets, please download them at the ../datasets/ folder in advance, and then run the following codes in your Python console:

```
import scipy.io

tensor = scipy.io.loadmat('../datasets/Guangzhou-data-set/tensor.mat')
tensor = tensor['tensor']
```

[2] https://github.com/xinychen/transdim

Python Implementation

Machine learning algorithms:

- Low-rank tensor completion
- Temporal matrix factorization

```python
def LRTC(dense_tensor, sparse_tensor, alpha, rho, theta, epsilon, maxiter):
    """Low-Rank Tensor Completion with Truncated Nuclear Norm, LRTC-TNN."""

    dim = np.array(sparse_tensor.shape)
    pos_missing = np.where(sparse_tensor == 0)
    pos_test = np.where((dense_tensor != 0) & (sparse_tensor == 0))
    dense_test = dense_tensor[pos_test]
    del dense_tensor

    X = np.zeros(np.insert(dim, 0, len(dim))) # \boldsymbol{\mathcal{X}}
    T = np.zeros(np.insert(dim, 0, len(dim))) # \boldsymbol{\mathcal{T}}
    Z = sparse_tensor.copy()
    last_tensor = sparse_tensor.copy()
    snorm = np.sqrt(np.sum(sparse_tensor ** 2))
    it = 0
    while True:
        rho = min(rho * 1.05, 1e5)
        for k in range(len(dim)):
            X[k] = mat2ten(svt_tnn(ten2mat(Z - T[k] / rho, k), alpha[k] / rho, np.int(np.ceil(
        Z[pos_missing] = np.mean(X + T / rho, axis = 0)[pos_missing]
        T = T + rho * (X - np.broadcast_to(Z, np.insert(dim, 0, len(dim))))
        tensor_hat = np.einsum('k, kmnt -> mnt', alpha, X)
        tol = np.sqrt(np.sum((tensor_hat - last_tensor) ** 2)) / snorm
        last_tensor = tensor_hat.copy()
        it += 1
        if (it + 1) % 50 == 0:
            print('Iter: {}'.format(it + 1))
            print('MAPE: {:.6}'.format(compute_mape(dense_test, tensor_hat[pos_test])))
            print('RMSE: {:.6}'.format(compute_rmse(dense_test, tensor_hat[pos_test])))
            print()
        if (tol < epsilon) or (it >= maxiter):
            break

    print('Imputation MAPE: {:.6}'.format(compute_mape(dense_test, tensor_hat[pos_test])))
    print('Imputation RMSE: {:.6}'.format(compute_rmse(dense_test, tensor_hat[pos_test])))
    print()

    return tensor_hat
```

- Define nonstationary temporal matrix factorization (`notmf`).

```python
def notmf(dense_mat, sparse_mat, rank, d, lmbda, rho, season, maxiter):
    dim1, dim2 = sparse_mat.shape
    W = 0.01 * np.random.randn(rank, dim1)
    X = 0.01 * np.random.randn(rank, dim2)
    A = 0.01 * np.random.randn(rank, d * rank)
    if np.isnan(sparse_mat).any() == False:
        ind = sparse_mat != 0
        pos_test = np.where((dense_mat != 0) & (sparse_mat == 0))
    elif np.isnan(sparse_mat).any() == True:
        pos_test = np.where((dense_mat != 0) & (np.isnan(sparse_mat)))
        ind = ~np.isnan(sparse_mat)
        sparse_mat[np.isnan(sparse_mat)] = 0
    dense_test = dense_mat[pos_test]
    del dense_mat
    Psi = generate_Psi(dim2, d, season)
    show_iter = 100
    temp = np.zeros((d * rank, dim2 - d - season))
    for it in range(maxiter):
        W = conj_grad_w(sparse_mat, ind, W, X, rho)
        X = conj_grad_x(sparse_mat, ind, W, X, A, Psi, d, lmbda, rho)
        for k in range(1, d + 1):
            temp[(k - 1) * rank : k * rank, :] = X @ Psi[k].T
        A = X @ Psi[0].T @ np.linalg.pinv(temp)
        mat_hat = W.T @ X
        if (it + 1) % show_iter == 0:
            temp_hat = mat_hat[pos_test]
            print('Iter: {}'.format(it + 1))
            print('MAPE: {:.6}'.format(compute_mape(dense_test, temp_hat)))
            print('RMSE: {:.6}'.format(compute_rmse(dense_test, temp_hat)))
            print()
    return mat_hat, W, X, A
```

Python Implementation

Easy to **switch from CPU to GPU**

NumPy $\Rightarrow$ CuPy

```
import numpy as np
```

```
import cupy as np
```

Only use `numpy`? Advantage:

- Fewer packages can improve the reproducibility

Post Something That Matters

Post well-documented data processing files (e.g., processing Chicago taxi data)

- Beginners to build coding skills
- Researchers to build research ideas

Matching Taxi Trips with Community Areas

There are three basic steps to follow for processing taxi trip data:

- Download taxi trips in 2022 in the `.csv` format, e.g., `Taxi_Trips_-_2022.csv`.
- Use the `pandas` package in Python to process the raw trip data.
- Match trip pickup/dropoff locations with boundaries of the community area.

```
import pandas as pd

data = pd.read_csv('Taxi_Trips_-_2022.csv')
data.head()
```

For each taxi trip, one can select some important information:

- `Trip Start Timestamp`: When the trip started, rounded to the nearest 15 minutes.
- `Trip Seconds`: Time of the trip in seconds.
- `Trip Miles`: Distance of the trip in miles.
- `Pickup Community Area`: The Community Area where the trip began. This column will be blank for locations ourside Chicago.
- `Dropoff Community Area`: The Community Area where the trip ended. This column will be blank for locations outside Chicago.

```
df = pd.DataFrame()
df['Trip Start Timestamp'] = data['Trip Start Timestamp']
df['Trip Seconds'] = data['Trip Seconds']
df['Trip Miles'] = data['Trip Miles']
df['Pickup Community Area'] = data['Pickup Community Area']
df['Dropoff Community Area'] = data['Dropoff Community Area']
del data
df
```

Figure 2 shows taxi pickup and dropoff trips (2022) on 77 community areas in the City of Chicago. Note that the average trip duration is **1207.75 seconds** and the average trip distance is **6.16 miles**.

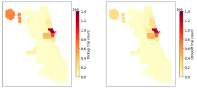


Figure 2. Taxi pickup and dropoff trips (2022) in the City of Chicago, USA. There are 4,763,961 remaining trips after the data processing.

For comparison, Figure 3 shows taxi pickup and dropoff trips (2019) on 77 community areas in the City of Chicago. Note that the average trip duration is **915.62 seconds** and the average trip distance is **3.93 miles**.

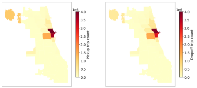


Figure 3. Taxi pickup and dropoff trips (2019) in the City of Chicago, USA. There are 12,484,572 remaining trips after the data processing. See the data processing codes.

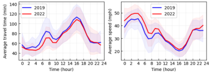


Figure 6. Average travel time and speed from area 32 (i.e., Downtown) to area 76 (i.e., Airport) in both 2019 and 2022.

```
import numpy as np
import matplotlib.pyplot as plt

fig = plt.figure(figsize = (8, 2.5))
ax = fig.add_subplot(1, 2, 1)
# Average travel time in 2019
m1 = df1.groupby(['hour'])['Trip Seconds'].mean().values / 30
s1 = df1.groupby(['hour'])['Trip Seconds'].std().values / 30
plt.plot(m1, color = 'blue', linewidth = 1.8, label = '2019')
upper = m1 + s1
lower = m1 - s1
x_bound = np.append(np.append(np.append(np.array([0, 0]), np.arange(0, 24)),
                    np.array([24 - 1, 24 - 1])), np.arange(24 - 1, -1,
y_bound = np.append(np.append(np.append(np.array([upper[0], lower[0]]), lower),
                    np.array([lower[-1], upper[-1]])), np.flip(upper))
plt.fill(x_bound, y_bound, color = 'blue', alpha = 0.05)

# Average travel time in 2022
m1 = df2.groupby(['hour'])['Trip Seconds'].mean().values / 30
s1 = df2.groupby(['hour'])['Trip Seconds'].std().values / 30
plt.plot(m1, color = 'red', linewidth = 1.8, label = '2022')
upper = m1 + s1
lower = m1 - s1
```

Source: https://spatiotemporal-data.github.io/Chicago-mobility/taxi-data

Post Something That Matters

Post scientific problems (e.g., spatiotemporal data modeling)

Optimizing Interpretable Time-Varying Autoregression with Orthogonal Constraints

Generally speaking, any spatiotemporal data in the form of a matrix can be written as $\boldsymbol{Y}\in\mathbb{R}^{N\times T}$ with N spatial areas/locations and T time steps. To discover interpretable spatial/temporal patterns, one can build a time-varying autoregression on the time snapshots $\boldsymbol{y}_1,\boldsymbol{y}_2,\ldots,\boldsymbol{y}_T\in\mathbb{R}^N$ (Chen et al., 2023). The time-varying coefficients in the autoregression allow one to characterize the time-varying system behavior, but the challenges still remain.

To capture interpretable modes/patterns, one can use tensor factorization formulas to parameterize the coefficients and the optimization problem can be easily built. However, a great challenge would be how to make the modes “more interpretable”, specifically, e.g., how to learn orthogonal modes in the modeling process. In this post, we present an optimization problem of the time-varying autoregression with orthogonal constraints as follows,

$$\begin{aligned}\min_{\boldsymbol{W},\boldsymbol{G},\boldsymbol{V},\boldsymbol{X}}\;&\frac{1}{2}\sum_{t=2}^{T}\left\|\boldsymbol{y}_t-\boldsymbol{W}\boldsymbol{G}(\boldsymbol{x}_t^\top\otimes\boldsymbol{V})^\top\boldsymbol{y}_{t-1}\right\|_2^2\\ \text{s.t.}\;&\begin{cases}\boldsymbol{W}^\top\boldsymbol{W}=\boldsymbol{I}_R\\ \boldsymbol{V}^\top\boldsymbol{V}=\boldsymbol{I}_R\\ \boldsymbol{X}^\top\boldsymbol{X}=\boldsymbol{I}_R\end{cases}\end{aligned}$$

where $\boldsymbol{W}\in\mathbb{R}^{N\times R}$ and $\boldsymbol{X}\in\mathbb{R}^{(T-1)\times R}$ refer to as the spatial modes and the temporal modes, respectively. This model can discover urban mobility transition patterns.

Source: https://spatiotemporal-data.github.io/probs/orth-var

Reproducibility Challenges

Resources

- Time / man power to create and maintain the material
- Storage (for data)
- Sustainable platforms

Creating and fostering a community

- Example of Traffic Intelligence, open source software for traffic video processing and safety analysis

The devil is in the details

- Have you tried to compare a new video object detector to the state of the art?

Thanks for your attention!

Any Questions?

About Nicolas Saunier:

- Homepage: https://nicolas.saunier.confins.net
- GitHub: https://github.com/nsaunier
- How to reach me: nicolas.saunier@polymtl.ca

About Xinyu Chen:

- Homepage: https://xinychen.github.io
- GitHub: https://github.com/xinychen
- How to reach me: chenxy346@gmail.com